

Program 7 - Random Singleness

Author: Abigail Pitcairn

Version: April 9, 2025

Code: <https://github.com/abbyPitcairn/BullDog>

Task 1: Building a First Singleton

I began by giving the AI the entire code from program 6 along with the instructions for task 1, since I was unsure of how to implement the changes myself and I was hoping I could get a starting point from the AI. ChatGPT stated that I had incorrectly implemented Referee, which was not true because a “Referee” was not yet implemented whatsoever, and it returned a simple Referee class.

I followed up by asking where in the code this new Referee class should go, based on the code I had provided. ChatGPT gave me the Referee class code again, and said this for “where to put the code”:

✓ Example Change in BulldogGameGUI.java

If your existing code looks like:

java

Copy

Edit

```
Referee referee = new Referee(game); // ✗ Don't do this anymore
referee.playGame(player1, player2, rounds, gameOutput);
```

Change it to:

java

Copy

Edit

```
Referee referee = Referee.getInstance(game); // ✓ Correct singleton usage
referee.playGame(player1, player2, rounds, gameOutput);
```

At this point, I was fearing that this task might take a while. ChatGPT was being very repetitive and was not fully correct on what it was repeating. It seemed like the AI was missing a key component of this task that was not explicitly written in the instructions, so I asked:

“Can you please separate the code from BulldogGUI so that there is a separate playGame() method in the Referee class that will instantiate a Singleton Bulldog Game?”

Now, ChatGPT went a bit off the rails. The beginning of its response was decent, stating our “goals”:

🧠 Goal

- `BulldogGameGUI` handles UI (buttons, inputs).
- `Referee` handles all game logic and game state.
- `Game` is created inside `Referee` (not in the GUI anymore).

However, the code it gave was very, very wrong, and did not look *anything* like the starting code I had provided just two prompts before. It rigidly declared exactly two players in the game setup method and

even had print statements in the BulldogGUI which were NOT there previously. None of it was usable. At this point, it had been about half an hour of prompting and reading responses.

I then began manually separating the game logic out to create a proper Referee; this involved making a lot of variables 'static' and ultimately broke everything.

When I asked the AI for help, it added a variable for the number of rounds in a game that was specified by the user (???), so instead of using that, ~~I threw my computer~~ I made these changes entirely myself.

```
try {
    int numRounds = Integer.parseInt(roundsText);
    if (numRounds <= 0) {
        throw new NumberFormatException();
    }

    Referee referee = Referee.getInstance();
    referee.playGame(name1, name2, numRounds, gameOutputTextArea);
} catch (NumberFormatException ex) {
    JOptionPane.showMessageDialog(BulldogGameGUI.this, "Number of rounds must be a positive integer.");
}
```

After lots of copying and pasting code to and from ChatGPT, I had a working game again with a Singleton Referee and Dice. This accomplished part of task 2 as well, because in order to have ChatGPT correctly implement the Dice initialization in playGame(), I specified that the Dice class was a Singleton design like it was in program 6, and it gave me the code for that, as well. I tested all of this together and it worked.

Total Task Time: Over two hours...

Task 2: Change the Dice Class

I started this task in task 1. My Dice class already used a Random object, so that did not need to be implemented. However, I now needed to change the Singleton design to apply to the Random object, not the Dice object.

I gave ChatGPT the instructions from task 2 and my current Dice class. It responded:

*"Got it! What you want is to **allow multiple Dice objects with different sizes** (like 6-sided or 2-sided dice), but have **all of them share a single, global Random instance** (which is the true Singleton in this case)."*

This initial response was very interesting to me, because I thought ChatGPT explained the task very well and it actually helped me understand a bit better what we (as in me and ChatGPT) were doing. The code itself was well done and came with Javadoc without me having to ask.

I asked ChatGPT to fix the playGame() method in the Referee class based on the new Dice class, which it did well on the first try. I also asked it to add a new 2-sided die to RandomPlayer, telling it explicitly what change to make and providing the current contents of the RandomPlayer class. This was also well implemented on the first response.

Some quick testing proved that the game functionality was still as expected, and I proceeded to task 3.

Total Task Time: 20 minutes

Task 3: Create an abstract RandomDice superclass

Since I use eclipse for this project, I was able to refactor -> extract superclass from the Dice class. This gave me a very simple superclass, RandomDice:

```
package Program7;
```

```

public class RandomDice {
    public RandomDice() {
        super();
    }
}

```

I gave ChatGPT the RandomDice class along with the explicit instructions for this task, and it made some modifications to RandomDice and Dice; specifically, it did the work of moving the “sides” parameter into the superclass and changing the subclass Dice to match the changes made to its new parent.

Total Task Time: five minutes

Task 4: Create a FakeRandom subclass of RandomDice

I created a class called FakeRandom and had it extend RandomDice in eclipse. From there, I semi-automatically generated the constructor and roll() method, although the roll() method was lacking. I prompted ChatGPT:

“I want to create a FakeRandom Dice object for testing purposes. I want to have a simple text file with strings of single digit numbers that I can use to test what happens when different sequences of numbers are rolled. Here is what I have so far:

```

package Program7;
public class FakeRandom extends RandomDice {
    public FakeRandom(int sides) { super(sides); }
    @Override
    public int roll() { return 0; } }

```

Please help me edit the roll() method so that it can read in a .txt file and return those values sequentially.”

It turns out that this implementation is a bit more complex than I had wanted it to be, so I revised the prompt to ask instead for the roll() method to take a String of integers and return those values sequentially when roll() was called.

```

package Program7;

public class FakeRandom extends RandomDice {
    private int[] sequence; // Array holding the sequence of numbers
    private int currentIndex; // Tracks the current index in the sequence

    public FakeRandom(int sides, int[] sequence) {
        super(sides);
        this.sequence = sequence;
        this.currentIndex = 0; // Initialize the index to start at the first element
    }

    @Override
    public int roll() {
        // If we've reached the end of the sequence, restart from the beginning
        if (currentIndex >= sequence.length) {
            currentIndex = 0; // Loop back to the start
        }

        int result = sequence[currentIndex]; // Get the number at the current index
        currentIndex++; // Move to the next number in the sequence

        return result; // Return the current number in the sequence
    }
}

```

I had to change the prompt a couple of times to get this, and then I manually changed the code to return -1 instead of restarting. Since this was solely for testing, it didn't make sense to have it restart.

Total Task Time: 15 minutes

Task 5: Create a SevenPlayer class

I copied FifteenPlayer and pasted it back into Program7, renamed the class to SevenPlayer, then using command-f, I was able to find everywhere it said "15/fifteen" and replaced it with "7/seven" automatically.

Total Task Time: one minute

Task 6: Create a FakeRandom test class

I began this task by creating a new JUnit test suite in my project in eclipse. I then gave ChatGPT the task 6 instructions, the automatically generated JUnit test suite, and the SevenPlayer class, explicitly stating what everything was that I was giving it.

It was at this point that I started getting errors with the type "Dice" in the play() method. I went back to the Player abstract class and manually changed the play() method to take the superclass type of RandomDice instead of only its subclass Dice. I then went into each Player class and changed the parameter of play() from Dice to RandomDice to get rid of all error messages.

As for ChatGPT's JUnit test class, it looked really good and made excellent use of the new FakeRandom dice. However, one test did not pass:

```
// Test case where the player's score exceeds 7 and stops at 7
@Test
public void testPlayerStopsAfterExceeding7() {
    // Simulate a sequence where the score exceeds 7 but stops at 7
    int[] rolls = {3, 3, 2}; // Rolls add up to more than 7 (total 8)
    dice = new FakeRandom(6, rolls);

    int score = player.play(dice);

    assertEquals(7, score); // Player should stop at 7, not exceed it
}
```

This is because while the SevenPlayer should stop here, the actual score is not 7, it should be 8. I simply changed the 7 to an 8 in the *assertEquals()* and then all tests passed.

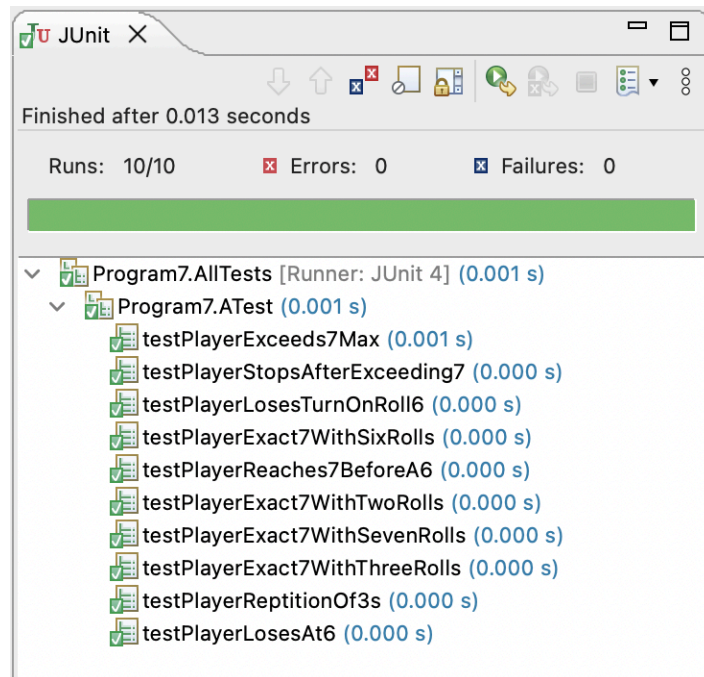
What was nice about the generated tests was that it included three cases that I needed:

1. When the first roll is 6 -> return 0
2. When the rolls add up to exactly 7 -> return 7
3. When the rolls add up to over 7 -> return total

However, there were only five total tests and three of them did essentially the same thing; return seven when two to three rolls added up to a total of seven. Using the outline provided by ChatGPT, I manually added some more boundary tests to match what was done in class.

I would also like to note that ChatGPT added way, way, way too many */*** comments and NO Javadoc. I gave it the test class again and asked it to add Javadoc and remove any */*** comments. While it did well with adding Javadoc, I had to manually remove the */*** comments which appeared on nearly every line.

Final result running AllTests test suite:



Total Task Time: 45 minutes

Task 7: Javadoc

Partially done during the previous task, I was able to efficiently copy/paste the classes that were missing documentation into ChatGPT and have it add the documentation for me.

Annoyingly, it also added green comments on random lines, which I manually removed.

Total Task Time (with proof-readin): 20 minutes

Summary

Total Assignment Time: 3 hours and 50 minutes

This assignment was not too difficult to complete with the help of ChatGPT, especially after overcoming the initial challenge of separating the game logic into a referee class. Originally, BulldogGameGUI had all the game logic, GUI functionality, and player initialization highly coupled. After separating the tasks myself, ChatGPT was able to format and fully integrate the changes across classes.

ChatGPT was also essential in creating the RandomDice and FakeRandom classes. While I generated some of this through Eclipse IDE, ChatGPT perfected it and seamlessly integrated it across classes, saving time on manual tracking. The AI also explained Dice functionality changes, helping me understand what to modify.

The easiest parts were creating the SevenPlayer class, which was simple, and generating Javadoc, which ChatGPT handled efficiently. Task 6, creating JUnit test cases, was surprisingly easy as well. ChatGPT produced reasonable test cases with minimal prompting. While they weren't extensive, they saved me time, and I could modify them for my own tests.

Overall, ChatGPT was invaluable to this project. If I had more time, I'd focus on improving the GUI's visual appeal, though that would take more time than AI can fix quickly.